Embedding Community-Specific Resource Managers in General-Purpose Grid Infrastructure

Ian Foster[1,2] Kate Keahey[1] Carl Kesselman[3] Erwin Laure[4]
Miron Livny[5] Stuart Martin[1] Mats Rynge[3] Gurmeet Singh[3]

[1] Math & Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A.
[2] Department of Computer Science, The University of Chicago, Chicago, IL 60615, U.S.A.
[3] Information Sciences Institute, U. Southern California, Marina del Rey, CA 90292, U.S.A.
[4] CERN, CH-1211 Geneva 23, Switzerland.
[5] Department of Computer Science, University of Wisconsin, Madison, WI 53706-1685, U.S.A.

Abstract

An important mode of Grid operation is one in which a community or (as we call it here) a *virtual organization* (VO) negotiates an allocation from a *resource provider* and then disperses that allocation across its members according to VO policy. Implementing this model requires that a VO be able to deploy and operate its own resource management services within the Grid. We argue that a mechanism that allows for the creation, and subsequent monitoring and control, of *managed computations* provides a simple yet flexible solution to this requirement. We present an architectural framework that addresses the security, policy specification, and policy enforcement concerns that arise in this context. We also describe an implementation based on Globus Toolkit and Condor components, and present performance results.

1 Introduction

The emergence of Grids has seen the creation of a uniform infrastructure for sharing resources across organizational boundaries. As production deployments become more common and user communities become larger, usage patterns are shifting towards the creation of multi-purpose Grids whose resources are consumed by one or more *virtual organizations* (VOs) [13]. Requirements for scalability and flexibility in management make it important that *resource providers* be able to allocate resources (e.g., nodes on a compute cluster, disk space on a storage system, or CPU fraction on a single processor) to VOs rather than to individual users. The VO may then disperse its assigned resources among its members as it sees fit: for example, giving “analysis” activities higher priority than “simulations,” or project leaders greater access rights than students. In effect, the resource provider delegates to a VO (or, more specifically, to VO administrators) the right to control the use of a resource allocation by members of that VO.

In such scenarios, we face the challenge of providing mechanisms by which the VO can manage its allocated resources: what VO members are allowed to use them, what tasks should be assigned to the available resources, and when should requested tasks be performed. The issues associated with VO resource management are similar to those that must be solved by each individual resource provider. However, the VO also has unique characteristics, due to its dynamic creation, adaptation over time, need to layer on an existing resource management framework and the need to federate decision making across multiple resources in a scalable way. These unique characteristics dictate that specialized mechanisms are required.

We are concerned here with identifying general mechanisms for deploying VO management solutions as part of VO establishment, for monitoring and controlling the execution of those mechanisms, and for managing interactions between VO decision making processes and the resources that they control. As we explain in Section 2, these mechanisms may be required to provide for the dynamic deployment of VO-specific components, the allocation of resources to those components to meet quality of service requirements, and the restart of those components following various forms of failure.

We propose an architectural framework and an implementation approach that address these requirements. The framework is based on a construct that we call the *managed computation*, a computational activity that a client can create with specified persistence properties and resource constraints, and then monitor and manage, via operations defined within a managed computation factory interface (Section 3). The framework also addresses security issues relating to delegation and credential management (Section 4).

Our implementation approach uses the Globus Toolkit version 4 (GT4) Web Services (WS) Grid Resource Allocation and Management (GRAM) service [9, 11] and the Condor resource manager [18, 19] within a service-oriented architecture, with Condor providing local monitoring and management functions and GRAM providing network access, security, policy callouts, and other related functions (Section 5).

We show in Section 6 how the managed computation construct can be used to implement a use case in which a "VO resource manager" deployed as a managed computation on a "service node" submits jobs to a "cluster," with the resources consumed by the VO resource manager constrained. Finally, we report on initial experimental results that evaluate our implementation from the perspectives of both performance and management effectiveness (Section 7), discuss related work (Section 8), and conclude (Section 9).

We summarize the principal contributions of this work as follows:

- A resource management architecture that allows for the dynamic deployment and subsequent management of VO-specific resource management logic, while addressing security, policy, and quality-of-service enforcement concerns.
- An implementation approach that exploits features of GT4 and Condor.
- Performance experiments that demonstrate that GT4 plus Condor can provide for effective enforcement of resource guarantees.

2 Resource Management for Virtual Organizations

Virtual organizations may be created and destroyed, with lifetimes varying from minutes to years. A VO's membership may vary during its lifetime, as may the resources available to its members to accomplish their work. These dynamics, as well as normal evolution of mission, priorities and requirements, mean that VO-level policy with respect to membership and resource consumption will inevitably change over time.

Such VO dynamics means that it is impractical for a VO-level resource management strategy to require the modification of policy at individual resource providers (for example, by creating VO-specific task submission queues or creating local user accounts) each time a VO adds a member or alters its operational policy. Such an approach would suffer from poor responsiveness and scalability, and would also preclude the federated management of multiple VO resources.

For these reasons, we advocate an alternative approach based on a separation of concerns between intra-VO resource management policy (e.g., “Ann has higher priority than Joe”) and resource provider policy towards a VO (e.g., “VO A has 30% of my CPU”). In this approach, resource providers are concerned solely with enforcing policies regarding VO access; intra-VO scheduling decisions are handled by separate VO-specific resource management infrastructures or *VO resource managers* (VORMs). Each VORM is responsible for arbitrating among requests issued by the users of its VO. Thus, we arrive at a two-level resource management scheme. End users issue requests to VORM(s), requesting that tasks be performed. If and when the VO policy indicates that a request is allowable, a VORM forwards the request to an appropriate resource provider for execution against the VO’s allocation.

The dynamic nature of VOs dictates that we be able to deploy and manage such VORMs *on demand* with no human intervention required. Parsimony in architectural features dictates that the VORM be treated identically to any other VO task. Thus, from the resource provider perspective, a VORM should be initiated and managed using standard resource management interfaces. The resource used by a VORM may be constrained, the VORM must be managed, and the VORM may be subject to policy enforcement, just like any other task managed by the resource. The interaction between VORM and resource provider should also not be specialized: a VORM should initiate VO tasks using the same standard resource management interfaces used to create the VORM. However, from the perspective of its VO, a VORM fulfills a vital *infrastructure* function and hence (in contrast to other VO tasks) it is important that the VORM be robust to transient failures so that they can provide a persistent capability to the VO.

The actual function of a VORM may be highly specialized to a particular VO, or it may provide generic behaviors such as simple batch scheduling. Our primary concern is not VORM behavior, but rather the creation of standard mechanisms by which these behaviors can be deployed onto an existing Grid-wide resource management framework.

3 The Managed Computation

We now turn to the question as to how to provide uniform mechanisms by which a VO can deploy and operate a VO specific resource management infrastructure. VO-specific management environments deployed in current production Grids are based on customized, hand-crafted deployments that cannot easily be ported to other environments. Furthermore, because they do not use common underlying Grid mechanisms, they cannot respond easily to the dynamics of the VO lifecycle. These considerations motivate the approach considered here, which is to use established Grid infrastructure mechanisms to address VO resource management requirements.

With this goal in mind, we propose to meet the requirements outlined in Section 2 via a single abstraction that we call a *managed computation*. A managed computation is one that we can start, stop, terminate, monitor, and/or control. It adheres to, and runs within, a local resource security, auditing, and accounting infrastructure. It is subject to policy enforcement, including resource provisioning agreements. As we discuss below, we define these management functions in terms of a *managed computation factory interface* that defines operations that a client can use to request the creation, monitoring, and control of managed computations. A *resource manager* controls the managed computation and enforces site policies. As we will see in Section 5, the existing widely used GRAM interface already implements a managed computation factory interface, and thus implementing this interface is straightforward.

This managed computation construct can be applied pervasively to the design of the VO resource management environment discussed above:

- A VORM runs on a physical resource provided by a resource owner. To control the resource consumption, the VORM is structured as a *computation managed by the resource provider*. As a managed computation, the VORM may also request that the manager ensure that it is restarted in the case of failure.
- The VORM accepts requests from its VO's user community. As it services these request, the VORM may need access to additional functions, such as job staging, data movement, and job submission. These functions can be provided to the community as additional managed computations.
- As part of its operations, the VORM may request that operations take place on other resources. Such requests can be structured as requests to other entities to create new managed computations, such as compute tasks on a cluster.

The important point to take away from this discussion is that the entire VO resource management architecture can be rendered in terms of the creation and operation of managed computations for different purposes. There is no need to introduce any other special architectural or implementation concepts. In brief:

- We view all computations as managed regardless of the resource on which they execute, be it (for example) a "service node" (for a VORM) or a "compute cluster" (for a task submitted via a VORM). Indeed, in the managed computation model, there is no distinction between the two.
- We use standard interfaces to managed computation factories to request the creation of managed computations and to control those computations subsequently, regardless of whether those computations are VORMS to be created for VOs or jobs to be run on computational clusters on behalf of a VORM (acting on behalf of its VO's users).

Managed computations are created by a class of network service that is generically referred to as a *managed computation factory.* As with any network service, the managed computation factory's behavior is defined completely in terms of its interface. This interface defines operations that a client can use to negotiate the following capabilities:

- *Deployment*: Creation of a managed computation on a computational resource.
- *Provisioning*: Specification of how many resources may be consumed by the computation, and in what way.

- *Persistence*: Specification of what to do if the environment in which the computation is executing, or the computation itself, fails.
- *Monitoring*: Client monitoring of managed computations.
- *Management*: Client control over computation lifetime.

A computation created by a managed computation factory may itself define a network interface and thus operate as a service, in which case we may refer to it as a *managed service*. In addition, a service created by a managed computation factory may itself be a managed computation factory. Indeed, this is a good way to think about a VORM, if its function is (as is often the case) to create computations on other resources for its VO.

Figure 1 illustrates the major components of a managed computation factory:

- *The factory service proper* provides the interface to the network, formatting and exchanging messages, and implements the overall control of managed computation factory function. The interface to the managed computation factory defines the functions available to a management client. These functions include the ability to create, configure, and destroy a managed computation.
- *A policy module* determines authorization, resource constraints, and persistence policy. In particular, the policy module is used to determine if a managed computation request (e.g., to create a VORM) is consistent with the operational policy of the resource provider that is to host the managed computation.
- *A resource manager*. This resource-specific management system is responsible for creating the managed computation, enforcing persistence and policy, and implementing management operations such as service termination or suspension. The managed computation factory interacts with the resource manager via manager-specific commands and protocols.

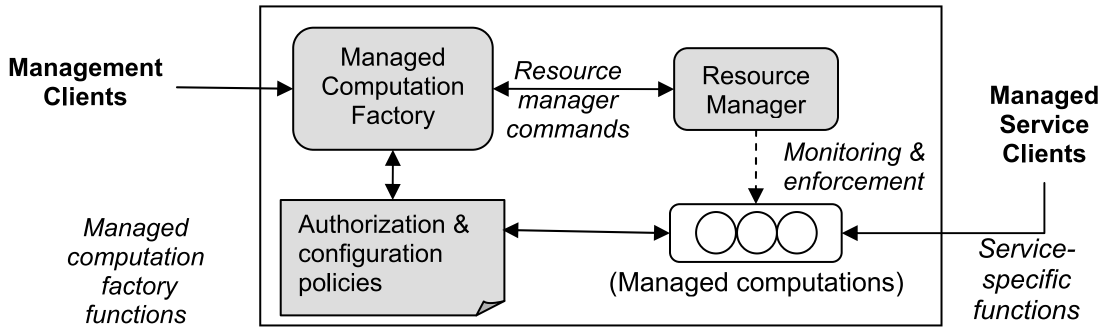
Management Clients
Managed Computation Factory
Resource manager commands
Resource Manager
Monitoring & enforcement
Managed Service Clients
Managed computation factory functions
Authorization & configuration policies
(Managed computations)
Service-specific functions

Figure 1: The managed computation factory

4 Security Issues in VO Scheduling

Five entities or principals can potentially participate when a VO user submits a request to a VORM to run a task on their behalf on some computational resource:

1. the *VO user* who submits tasks to be scheduled,
2. the *VORM resource provider* who hosts the VORM,
3. the *VO administrator* who initiates the creation and operation of the VORM,

4. the *task resource provider* who hosts the tasks requested by the VO user, and
5. the *VO resource consumer* who submits the job to the task resource provider,

Techniques for establishing the identity of participants in a distributed Grid environment such as the Grid security infrastructure [12] are well established and widely deployed [4]. Approaches by which finer grain policy decisions may be made via attribute assertions [16] are becoming more widespread [5, 27]. The use of attributes (or roles) in addition to identities can provide for greater flexibility and scalability in the resulting environment, and may be especially useful in identifying VO users as well as distinguishing which members may fill roles such as VO administrator and resource consumer.

A VORM is initiated by a VO administrator. The VO administrator must (mutually) authenticate with the VORM resource provider, providing any attributes necessary to identify the user in the VO administrator role. (Because the allocation of resources to the VO is an agreement between the resource provider and the VO, it is ultimately up to the resource provider to determine which attributes may be required.) Once authenticated, the VO administrator can initiate a managed task. This task runs with the rights of the VO administrator, the most important of which is the ability to consume resources allocated to the VO on the scheduler resource provider.

In order to submit a task to a VORM, VO users must identify themselves to the VORM. The same mutual authentication methods used for authenticating the VO administrator to the resource provider may be used for this purpose. Based on this authentication and any associated attributes asserted by the user or other sources, the VORM may apply policy to determine if the requestor is in fact currently a VO member, and if so, what scheduling policy should be assigned to the requested task.

If policy indicates that the VO user's task request should be executed, then task submission to the task resource provider can occur via one of two methods:

1. The VORM can use the managed computation creation interfaces to cause the schedulable task to execute on a target resource.
2. The VORM can instead not submit the task to the target resource itself, but rather grant the user permission to submit the task directly.

In the first case, the VORM submits the task under the identity of a VO resource consumer using credentials provided to the VORM when it was created. (See Section 5.1 for how this task can be done using the GT4 GRAM service.) The advantage of this approach is that VO membership is determined only by the VORM; members may be added or deleted without involving the target resource provider. A disadvantage is that the use of VO resource consumer credentials can complicate data staging operations in which data owned by the submitting user is transferred to the target resource. In non-VO deployments, the user simply delegates to the target resource user credentials that may be used to access remote data. However, if (as here) the target resource does not have the user's credentials, then alternative mechanisms are needed.

In practice, the resource provider may require the identity of the requesting user for auditing purposes, or to apply additional policy such as black list enforcement. For this reason, the resource provider may require that the VORM provide the identity of the original user as part of the request.

In the second approach listed above, the VORM acts in a manner similar to a community authorization service [22] or SHARP [15], providing the user with a cryptographically signed credential authorizing the user to consume resources on the task resource provider on behalf of the VO. Because the user must authenticate to the resource directly, this approach may be desirable when it is important to know the identity of the requesting VO user and the resource provider does not trust the VO to report this identity accurately. This approach also eliminates data staging complexities. However, this approach places a greater burden on submitting users, and precludes certain scheduling strategies, such as those in which the VORM queues jobs or manages submitted jobs.

5 A Managed Computation Framework Implementation

We describe here how a managed computation framework that meets our requirements can be implemented by using a combination of the *GT4 GRAM service* and *Condor resource manager*. In brief, we use GT4 GRAM as the managed computation factory service and Condor as the local resource manager.

5.1 GT4 GRAM Service as a Factory

GT4 WS GRAM is a Web Service that implements operations for creating—and subsequently monitoring and controlling—computations specified by a Job Description Schema. The interface uses operations standardized in the Web Services Resource Framework and WS-Notification [10] specifications for such purposes as monitoring, notification, and lifetime management. In addition, a GGF-standard authorization callout allows for flexible authorization policy [29]. GRAM includes the use of a *delegation service* [28], which allows the initiator of a managed computation to enable the resulting computation to perform actions on behalf of the initiator or any other principal that has provided delegated credentials.

GT4 WS GRAM applies policy and then dispatches approved requests to a local resource management system for local scheduling and policy enforcement. A wide range of resource managers, such as PBS, SGE, LSF, or Condor can be used, depending on the target environment. In a clustered environment, these managers typically operate in a space-shared mode, providing tasks with exclusive access to a resource. However, GRAM can also be used to deploy and manage computations onto non-space shared systems. In such systems, GRAM is often configured to use a best effort resource management strategy (GRAM's "fork gatekeeper") by which jobs are simply forked and scheduled by the operating system. However, shared resources can also be managed by a resource manager to provide more fine-grained policy and enforcement. We explore here the use of Condor as a local resource manager. Thus, while GRAM is sometimes viewed simply as a "job submission service," i.e., as a service for submitting jobs to batch schedulers, it more properly viewed as a general-purpose managed computation factory.

5.2 Condor as a Resource Manager

Condor provides mechanisms for scheduling tasks onto a pool of resources. It takes descriptions of required and offered resources and schedules them via a matchmaking mechanism [23]. Condor can be configured to offer both space sharing resource management in which only one task is matched to a resource at a time, or by using a virtual processor abstraction be configured to enable multiple tasks to be scheduled to a processor. In both cases, tasks initiated by Condor are monitored to ensure that policy requirements of the resource provider are enforced on a per (virtual) machine basis.

When exclusive access is offered, the policy tends to be simple, stating when a task can be mapped to the resource, how long it can run, and under what conditions the task may be terminated. The situation tends to be more complex in the time shared mode, when more then one task can be allocated to a resource. In this case the policy may specify both the maximum number of managed computations that can execute at one time and the maximum load that any one computation can apply to the node on which it executes. A request to create a computation is delayed or denied if the maximum number of computations has been reached; a computation that generates more than the maximum load is suspended or terminated. (See Section 7 for more details on the configuration.)

5.3 Persistence

Infrastructure services are those services that are critical to the operation of an organization. For example, GRAM may be critical to a resource provider, and a VORM to a VO. In the present context, we need to be able to ensure that the managed computation factory (GRAM), associated resource manager (e.g., Condor), and any managed computations (e.g., VORMs) persist beyond failure.

Both GT4 WS GRAM and Condor are themselves fault-tolerant, meaning that they can reinitialize their state from stable storage when restarted. Thus, if the platform on which GRAM and Condor execute should fail, both systems can be restarted and will reconnect themselves to all their active and submitted tasks. Beyond these capabilities, we need the ability to cause infrastructure services being managed by the resource manager to be restarted. Condor provides this capability, However, restart places responsibility on the managed computation, requiring it to checkpoint its state; thus the Condor mechanism cannot be applied blindly. In addition, persistence properties may be driven by policy statements that specify, for example, how often a service should be restarted after specific types of failures. Thus, we enable automatic restart for a managed computation only if requested by including a PERSISTENCE option in the job description provided to GRAM. This request is passed on to the local resource manager.

5.4 Implementing VORMs

One reason for deploying a VORM is to enable more scalable mechanisms for enforcing VO policy. However, a second motivation can be to create a different VO-specific resource management regime within the general-purpose Grid infrastructure. We illustrate this point by describing two alternative VO scheduling implementations. Of course, there are many other possibilities beyond the two described here.

5.4.1 A Condor-Based VORM

Condor provides a scheduling regime based on a ranked matching of task requirements to resource capabilities. At the heart of this system is a scheduling deamon (`schedd`) to which a user submits job requests, which are subsequently matched to, and executed on appropriate execution resources. Condor includes a version of `schedd`, called Condor-C, that can accept jobs from other `schedds` and be deployed dynamically [26].

To realize this scenario, we use our managed computation factory service to initiate a `schedd`, configured to register into a VO-wide Condor pool. A VO user can then submit a job to their local Condor pool, and the job will flow from that pool to the dynamically deployed `schedd` using Condor-C. The `schedd` submits the job to the computational resource for eventual execution using existing Condor mechanisms for accessing GRAM-mediated resources. We explore the performance of this scenario in Section 7.

Note that as we also use Condor in our managed computation factory service implementation, this use of Condor as a VORM means that we have two separate Condor deployments that are used in distinct ways. First, as described in Section 5.2, we use Condor as a resource manager for managed computations, such as VORMs. Second, we use Condor as a VORM implementation. The latter completely separate Condor deployment has no connection whatsoever with the Condor used as resource manager.

5.4.2 A GRAM-Based VORM

An alternative approach to VO resource management uses a GRAM-style VORM. Here, we use the managed computation interface to deploy a new GRAM service, which we configure according to VO policy requirements. The result is a management service with a GRAM interface that the VO user can use to submit jobs. The VO GRAM service can then in turn use a GRAM interface to consume resources offered by the target resource provider. This embedded GRAM approach has the advantage that VO applications written to the GRAM interface can operate in the VO environment without modification: in effect, what we achieve is a transparent virtualization of the underlying infrastructure into the VO space.

6 Deployment Scenarios

The use of standard remote task deployment and management interfaces for deployment of VORMs and for connecting VORMs to underlying resource providers gives us great flexibility with respect to where VORMs and user tasks are placed. For example, we often wish to place a VORM that is associated with a single target resource near that resource—perhaps on a *service node* (or, as it sometime called, a head node) associated with a computational cluster, or even on the target resource. In other configurations, a VORM may be associated with (and thus may select from among) more than one target resource. In such cases, we may wish to either collocate the VORM with one target resource, or place it on yet another resource that is “near” the target resources.

The beauty of the service-oriented approach is that no changes are required to our system implementation to accommodate these different configurations. Indeed, our use of standard interfaces means that a VORM need not create managed computations on an actual piece of physical hardware, but can instead interact with another nested VORM. Thus, a VO can blend strategies: collocating VORMs close to the target resource for reasons of performance and robustness, and using a higher-level VORM to select between alternative resources.

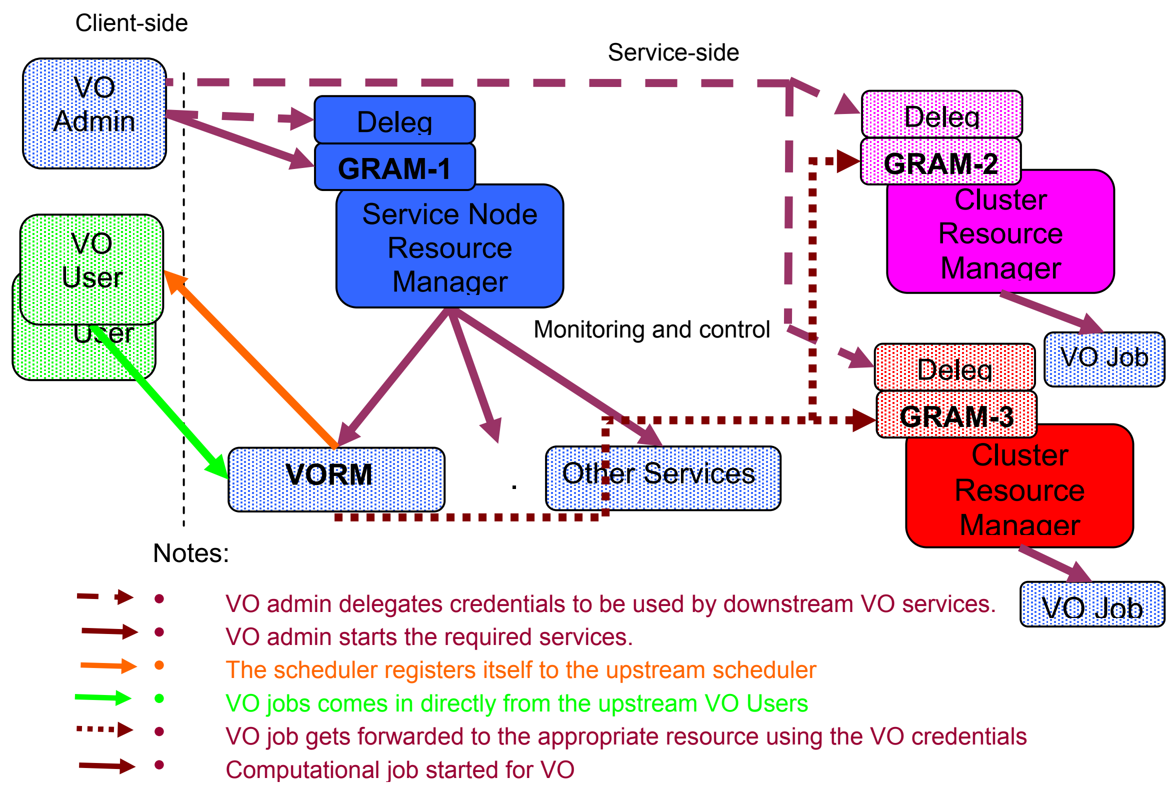
Client-side
Service-side
VO Admin
Deleg
GRAM-1
Service Node Resource Manager
VO User
User
Monitoring and control
VORM
Other Services
Deleg
GRAM-2
Cluster Resource Manager
VO Job
Deleg
GRAM-3
Cluster Resource Manager
VO Job
Notes:
VO admin delegates credentials to be used by downstream VO services.
VO admin starts the required services.
The scheduler registers itself to the upstream scheduler
VO jobs comes in directly from the upstream VO Users
VO job gets forwarded to the appropriate resource using the VO credentials
Computational job started for VO

Figure 2: VORM deployment scenario with one "service node" and two remote compute clusters

We show in Figure 2 one possible scenario in which we have one managed computation factory, GRAM-1, used to create VORMs, and two managed computation factories, GRAM-2 and GRAM-3, used to create computational tasks on attached clusters. The three GRAMs may be located on the same or different computer: if GRAM-1 is located on a service node associated with a cluster, and GRAM-2 and GRAM-3 provide access to that cluster, then the three GRAMs may be located on the same computer. Regardless of the precise configuration, the interactions proceed as follows:

1. The VO administrator uses the delegation service interface to make delegated credentials for the VO available to the GRAM services managing the service node and the computational cluster.
2. The VO administrator uses the GRAM interface to request that a VORM be created on the service node. This request may specify desired restart policy, and includes a reference to the delegation service on the compute nodes to which the VORM will be submitting jobs. The VO administrator is returned a handle with which it can monitor and control the new VORM. The service node resource manager enforces site policies relating to resource consumption (e.g., how much CPU and memory the VORM may consume) and may restart the VORM if the node fails and suspend or kill the VORM if site policies concerning resource consumption are violated.
3. A VO user authenticates to the VORM using their own credentials and submits jobs using a VORM-specific submission protocol such as Condor or GRAM. The VORM is responsible for queuing the job.

4. The VORM selects a job to execute and performs any setup that may be required, such as staging files. Any activity that the VORM performs on the service node is constrained by the policy enforced by the service node resource manager. Any operations performed by the VORM on remote nodes is constrained by the resource managers (if any) operating on those nodes.
5. The VORM uses GRAM to submit the VO job to the compute cluster. The job is submitted with VO or user credentials, depending on local policy, and is executed under a local account as determined by whatever mapping functions have been put in place by the resource owner (e.g., mapfile, dynamic account creation). The VORM uses the compute cluster's delegation service to delegate credentials to the job.

7 Experimental Studies

We conducted experiments to evaluate the suitability of our architecture and implementation when a VORM runs on a shared service node (or head node) that acts as a submit point for an associated computational cluster. GT4 WS GRAM on the service node provides two managed computation factories, one for computations on the service node and one for computations on the associated cluster. As described above, Condor is used as the resource manager for computations running on the service node. PBS is used to manage computations running on the cluster.

We used a Condor-C scheduler [26] as the VORM, as described in Section 5.2; this component is deployed dynamically using the GT4 GRAM computation factory interface. Note that this Condor-C VORM is completely independent of the Condor environment used to control the managed computations on the service node. We ran experiments on dual 2.2Ghz Intel Xeon machines running Red Hat 9 and Debian Sarge. Each machine had 1 GB RAM and a gigabit network connection.

In order to understand the effectiveness of managed computations for VO resource management, we measured how end-to-end VO job throughput varied with competing load from other activities on the service node. Figure 3 summarizes the results of this experiment, showing job throughput achieved by a Condor-C VORM process in both managed and unmanaged scenarios, as it repeatedly executed a trivial job on the computational cluster in the presence of increasing load. The load was caused by "misbehaving" processes, i.e., processes responsible for contributing to system load in excess of a prescribed limit. We see that, as expected, the performance in the unmanaged case collapses as the number of misbehaving jobs increases. However, in the managed cases we are able to prevent complete performance collapse by applying different management policies.

We experimented with two such policies. The first policy, applied to all computations on the service node, was as follows: "if the system load exceeds 10 and a managed computation's contribution to this load exceeds a prescribed threshold [a load of 3 for these experiments], then suspend the computation until the load is below the threshold". As can be seen in the figure while this policy certainly introduced some overhead, it also lead to a much more graceful performance degradation. A better policy proved to be to additionally impose a penalty of at least 10 minutes suspension on all misbehaving processes. In this case, the degradation was much better controlled although still observable as the load increased.

This shows that in practice the types of policy that can be applied – as well as the extent to which they can be effective – will strongly depend on the tools used to enforce them. In our case, a stricter policy compensated for the coarse-grain measurements used for load calculation as well as coarse-grain enforcement mechanisms. We also note that these mechanisms are not sufficient for tight levels of control: in the best case job throughput rate varies from nine jobs per second to slightly less then five. Alternative local resource managers such as hypervisor based solutions may be able to reduce this variability.

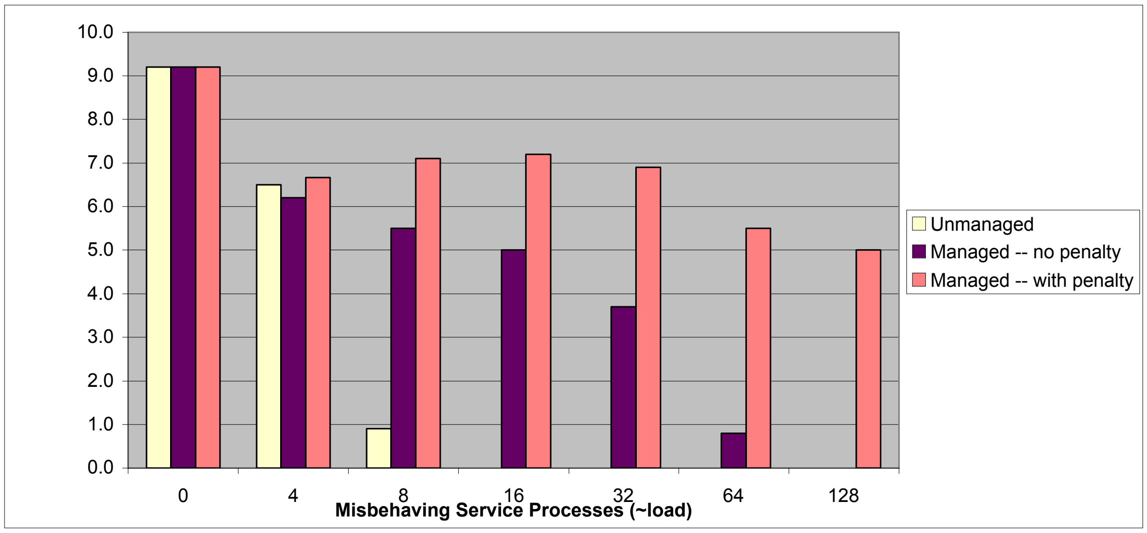

Figure 3: Throughput for a Condor-C VORM under load, when managed and unmanaged

As a baseline, we can compare the results of the managed, end-to-end solution with the throughput achieved when the same trivial jobs are submitted directly to a GRAM-mediated computational factory on the cluster. In this case, we see throughputs of approximately 10 jobs per minute. We conclude that the use of VORMs as managed computations need not negatively impact the throughput performance of the overall job management solution. In addition, we observe that the proposed solution can be extended easily to support multiple service nodes and multiple computational factories for computational resources. Hence, deployments can be engineered that can provide significantly higher levels of performance. (GT4 GRAM performance can also be improved: in a somewhat different configuration, we achieve 70 jobs per minute.)

8 Related Work

Many distributed system architectures and deployments separate management concerns between resource provider and VO in various ways. The concept of a "resource broker" is old and widespread [2, 20, 21]. Dumitrescu and Foster [8] describe and evaluate an architecture in which VO resource managers interact with resource providers on behalf of VO users. However, they focus on VO resource management strategies, not the managed service implementation issues considered here. Thain and Livny [26] use GRAM to deploy Condor agents directly to computational nodes, an approach that can provide significant performance benefits [25]. This “glide in” technique can be viewed as a special case of our more general approach.

At the interface and implementation level, PlanetLab [3] provides a programmatic interface for allocating execution environments (“slices”) on distributed computers, but not for managing the physical resources allocated to those environments or for starting or managing computations in those environments (other than via SSH). InVigo [1] provides for the managed deployment of virtual machines; however, the focus is on deploying applications not VO services.

The Xenoserver design [14] addresses the deployment and management of services and, as the authors point out, can be used to realize the concepts described here. Keahey et al. [17] describe a virtual workspace abstraction that is yet more general than our managed computation, supporting the configuration of arbitrary virtual machines. The GRAM design described here has evolved from an early job submission interface [6], influenced by thoughts on resource and service management [7].

As discussed in Section 4, capability-based systems such as CAS [22] and SHARP [15] represent an alternative authorization approach, in which a client obtains a ticket from a broker and then presents that ticket to a resource provider.

9 Summary and Next Steps

We have described an approach to deploying community-specific resource management logic (“Virtual Organization Resource Managers,” or VORMs) within a distributed computing infrastructure. Our approach addresses, at the architecture and implementation levels, the security and management issues required for this deployment to occur in a secure, robust, and performant manner.

From an architectural perspective, our approach has the advantage of being a pure service-oriented architecture. Individual components can be composed easily into higher-level components and configured into a wide variety of physical deployment strategies. Thus, we can easily deploy VORMs onto arbitrary service nodes or onto cluster nodes, create VORMs near or far from the resources to which they provide access, and build VORMs that schedule across multiple computing resources.

Our implementation approach is also elegant. In particular, it has the advantage of leveraging existing GT4 and Condor components unmodified, with the sole exception of a job description extension to specify persistence. In addition, our implementation can leverage the GT4 security implementation without modification, including GSI, delegation service, and interfaces to SAML-based policy processing engines.

In future work, we wish to explore the use of virtual machine technology [24] to provide finer-grained control over policy enforcement on managed computations. (While Condor’s widespread use and ease of configuration are advantages for a resource manager, the fact that it builds on traditional operating system mechanisms limits the degree of control and policy enforcement it can apply to shared resources.) We will also explore demand-driven provisioning of service nodes to meet user demand and support for management of distributed services.

Acknowledgments

We are grateful to fellow EGEE Design Team members for many helpful discussions on these topics. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, by the National Science Foundation (NSF) under its NSF Middleware Initiative, and by the European Commission program INFSO-RI-508833 through the EGEE project. Foster and Kesselman are also co-founders of Univa Corporation.

References

1. Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L. and Zhu, X. From Virtualized Resources to Virtual Computing Grids: The In-VIGO System. *Future Generation Computer Systems*. 2004.
2. Aloisio, G. and Cafaro, M. Web-Based Access to the Grid Using the Grid Resource Broker Portal. *Concurrency and Computation: Practice and Experience, Special Issue on Grid Computing Environments*, *13-14*. 2002.
3. Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T. and Wawrzoniak, M., Operating System Support for Planetary-Scale Services. *1st Symposium on Network Systems Design and Implementation*, 2004, 253-266.
4. Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J. and Welch, V. A National-Scale Authentication Infrastructure. *IEEE Computer*, *33* (12). 60-66. 2000.
5. Cornwall, L. and others Authentication and Authorization Mechanisms for Multi-Domain Grid Environments. *Journal of Grid Computing*, *2* (4). 301-311. 2004.
6. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S., A Resource Management Architecture for Metacomputing Systems. *4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, Springer-Verlag, 62-82.
7. Czajkowski, K., Foster, I. and Kesselman, C. Resource and Service Management. *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, 2004.
8. Dumitrescu, C. and Foster, I., Usage Policy-based CPU Sharing in Virtual Organizations. *5th International Workshop in Grid Computing*, 2004.
9. Foster, I., Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP International Conference on Network and Parallel Computing*, 2005, Springer-Verlag LNCS 3779, 2-13.
10. Foster, I., Czajkowski, K., Ferguson, D., Frey, J., Graham, S., Maguire, T., Snelling, D. and Tuecke, S. Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF. *Proceedings of the IEEE*, *93* (3). 604-612. 2005.
11. Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, *11* (2). 115-129. 1998.
12. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S., A Security Architecture for Computational Grids. *5th ACM Conference on Computer and Communications Security*, 1998, 83-91.

13. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, *15* (3). 200-222. 2001.
14. Fraser, K.A., Hand, S.M., Harris, T.L., Leslie, I.M. and Pratt, I.A. The Xenoserver Computing Infrastructure. University of Cambridge Computer Laboratory, Technical Report UCAM-CL-TR-552, 2003.
15. Fu, Y., Chase, J., Chun, B., Schwab, S. and Vahdat, A., SHARP: An Architecture for Secure Resource Peering. *19th ACM Symposium on Operating Systems Principles*, 2003.
16. Hughes, J. and Maler, E. Technical Overview of the OASIS Security Assertion Markup Language (SAML) v1.1, http://www.oasis-open.org/committees/security, 2004.
17. Keahey, K., Doering, K. and Foster, I., From Sandbox to Playground: Dynamic Virtual Environments in the Grid. *5th International Workshop in Grid Computing*, 2004.
18. Litzkow, M., Livny, M. and Mutka, M. Condor - A Hunter of Idle Workstations. *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, 104-111.
19. Livny, M. High-Throughput Resource Management. Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 311-337.
20. Nahrstedt, K. and Smith, J.M. The QoS Broker. *IEEE Multimedia*, *2* (1). 53-67. 1995.
21. OMG Common Object Request Broker: Architecture and Specification. 1991.
22. Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S., A Community Authorization Service for Group Collaboration. *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.
23. Raman, R., Livny, M. and Solomon, M. Matchmaking: An Extensible Framework for Distributed Resource Management. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, *2*. 129-138. 1999.
24. Rosenblum, M. and Garfinkel, T. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer* (May). 39-47. 2005.
25. Singh, G., Kesselman, C. and Deelman, E. Optimizing Grid-Based Workflow Execution. *Journal of Grid Computing* (To appear).
26. Thain, D. and Livny, M. Building Reliable Clients and Services. *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, Morgan Kaufmann, 2004.
27. Welch, V. Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective, 2004. http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf.
28. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L. and Tuecke, S., Security for Grid Services. *12th IEEE International Symposium on High Performance Distributed Computing*, 2003.
29. Welch, V., Siebenlist, F., Meder, S. and Pearlman, L. Use of SAML for OGSA Authorization. Global Grid Forum, Draft, 2003. www.globus.org/ogsa/security.